

Installable

PWAs

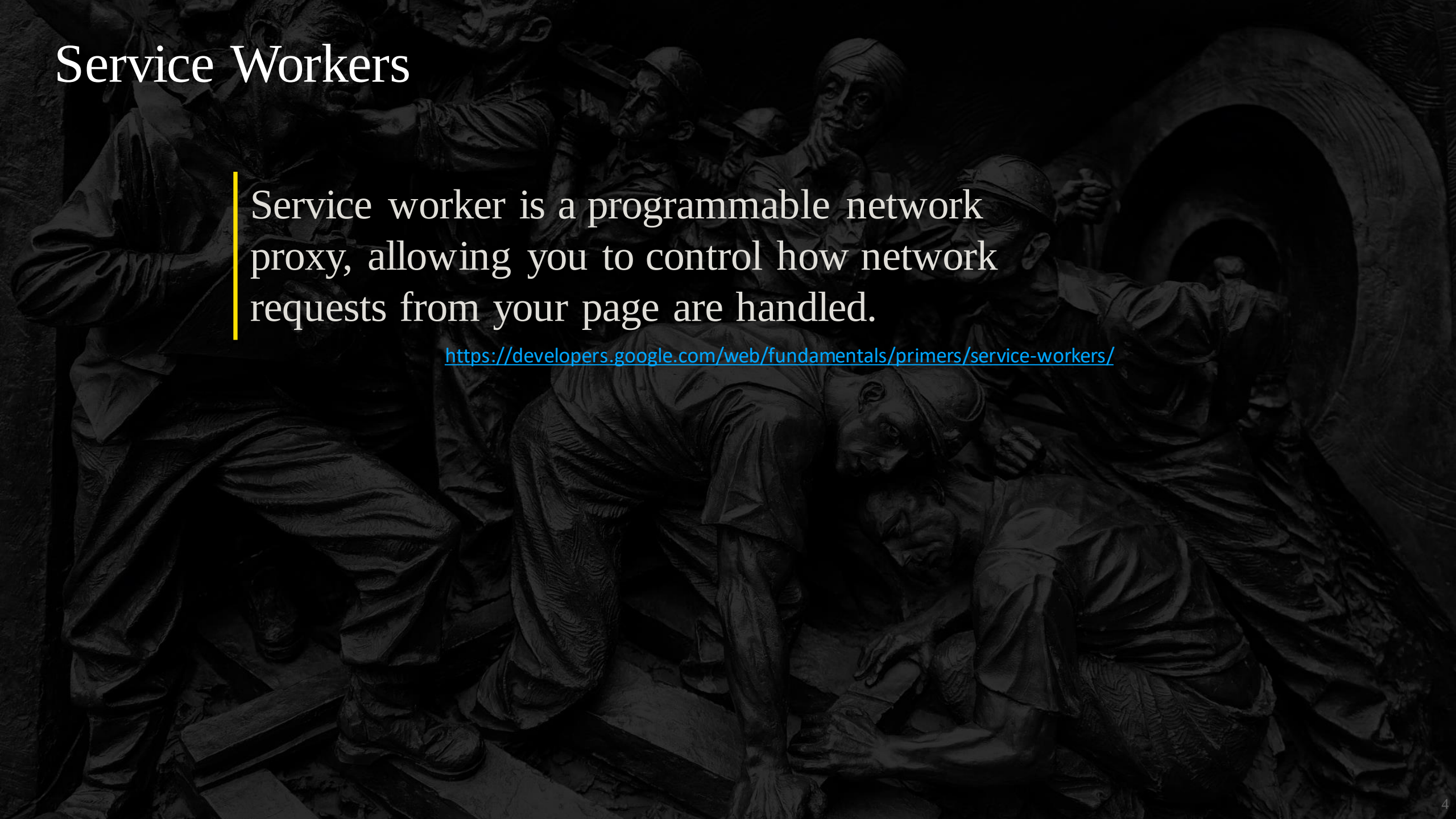


Detta verk är licensierad under en
[Creative Commons Erkännande 4.0](https://creativecommons.org/licenses/by/4.0/)
[Internationell licens](https://creativecommons.org/licenses/by/4.0/).

Content

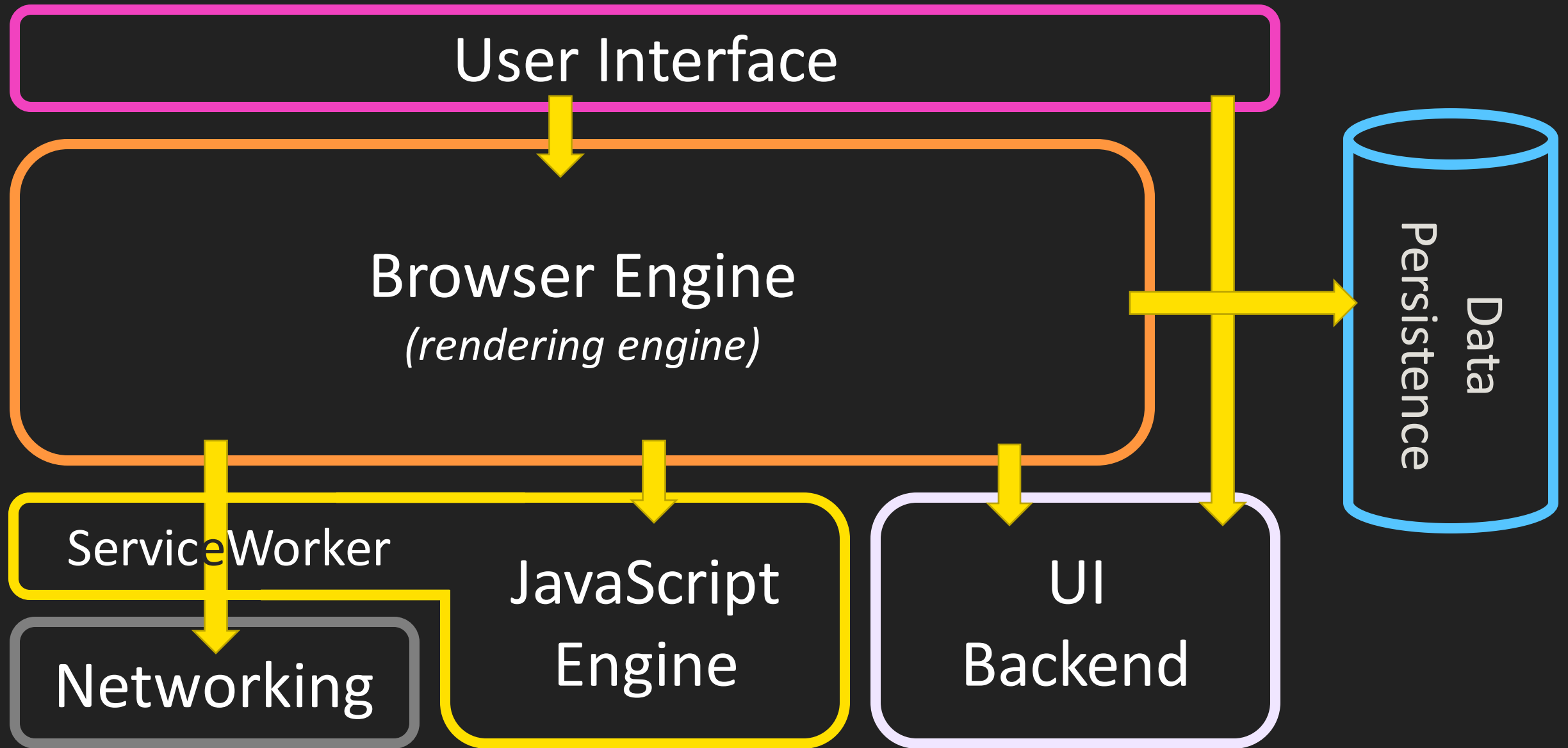
- ✓ Service Workers
- ✓ CachStorage
- ✓ WebManifest

Service Workers



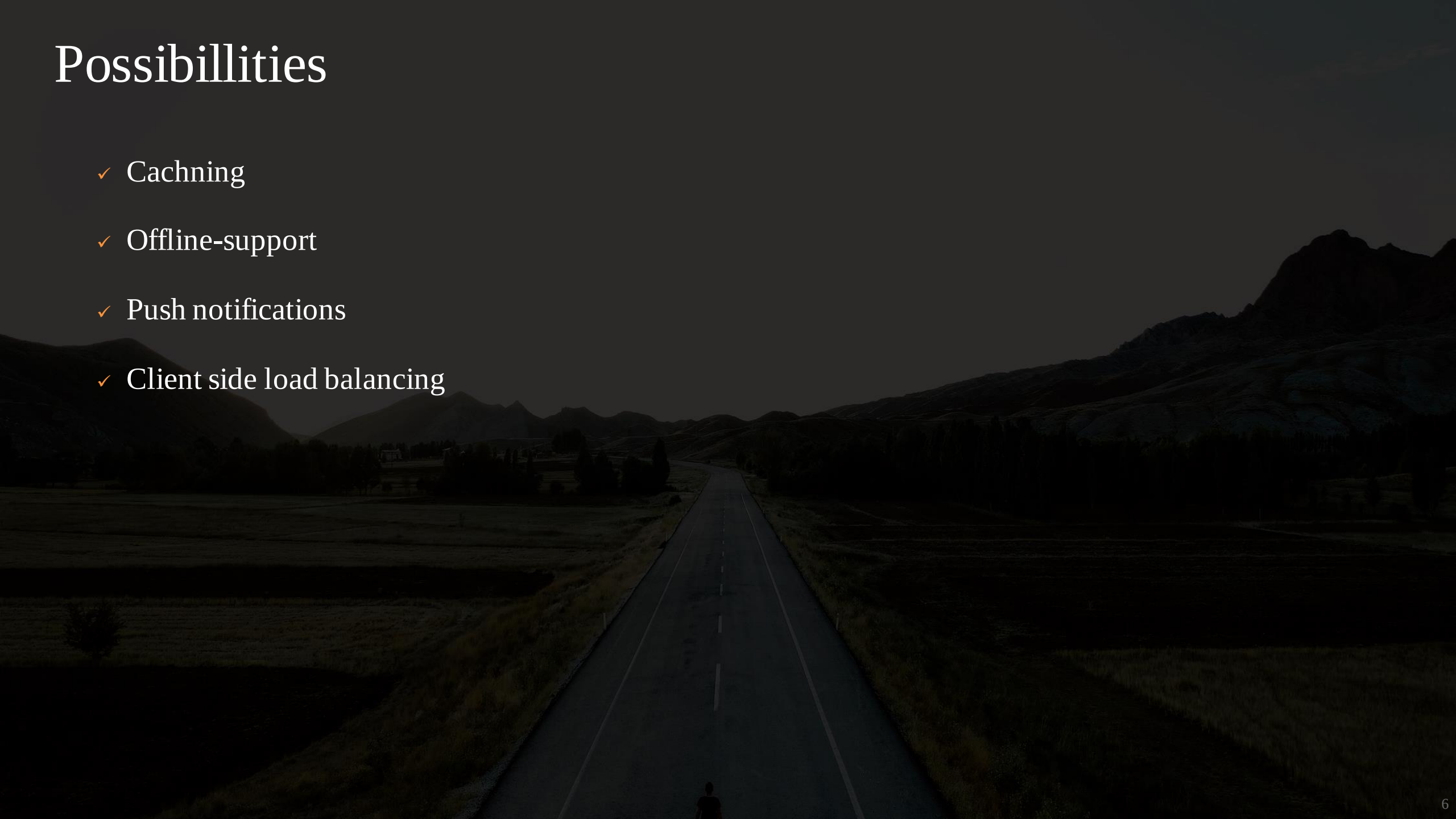
Service worker is a programmable network proxy, allowing you to control how network requests from your page are handled.

<https://developers.google.com/web/fundamentals/primers/service-workers/>



Possibilities

- ✓ Caching
- ✓ Offline-support
- ✓ Push notifications
- ✓ Client side load balancing



Requirements

- ✓ HTTPS (not on localhost)
- ✓ Only asynchronous calls are permitted
- ✓ Proficiency in promises
- ✓ Modern Browser
- ✓ Uses a separate global scope
(ServiceWorkerGlobalScope:WorkerGlobalScope)

Scopes

- ✓ window
- ✓ WorkerGlobalScope
- ✓ ServiceWorkerGlobalScope
- ✓ self

ServiceWorkers

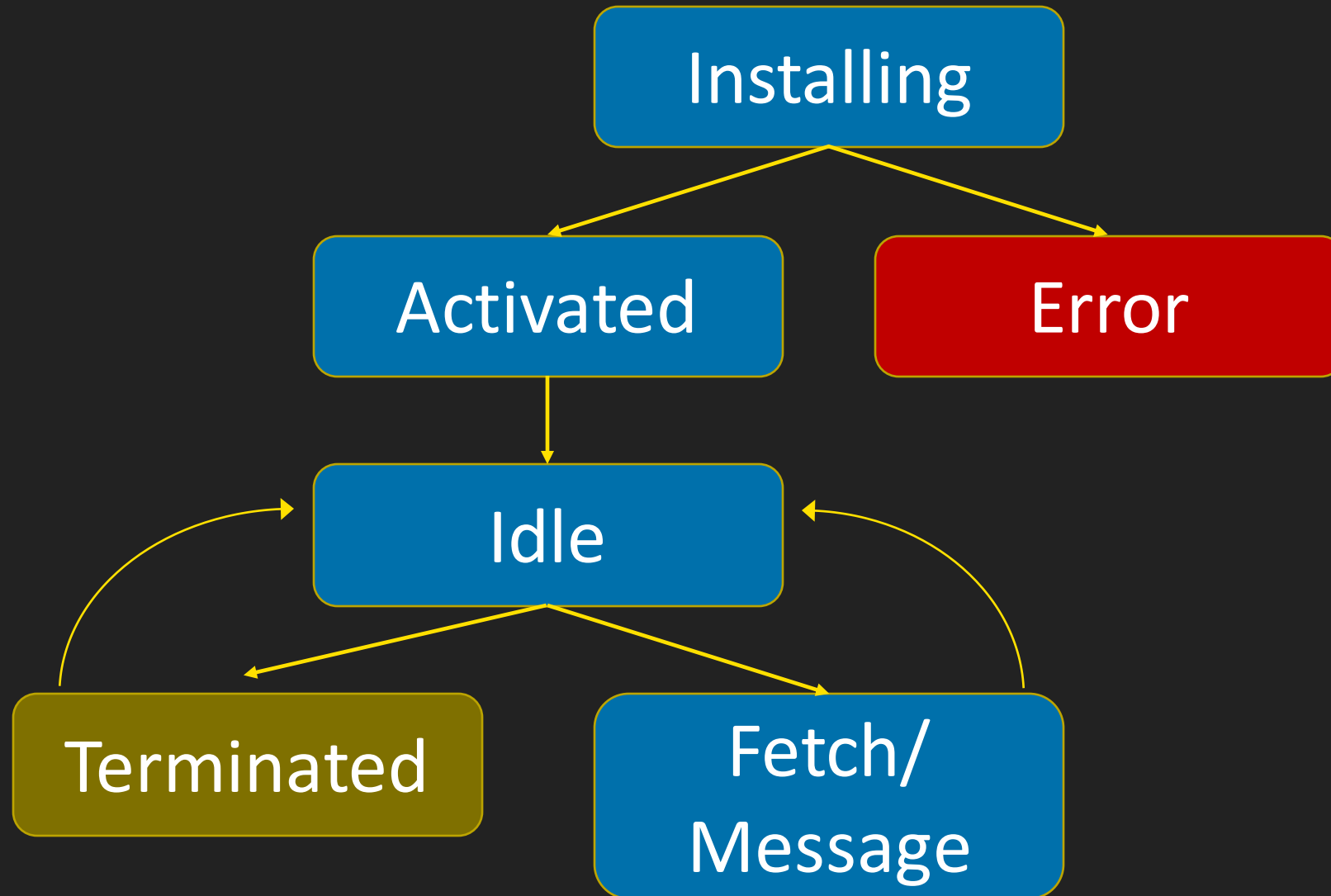


- ✓ CacheStorage-API
- ✓ Fetch-API
- ✓ IndexedDB
- ✓ WebSockets
- ✓ navigator/location
- ✓ setTimeout/setInterval
- ✓ (JSON)

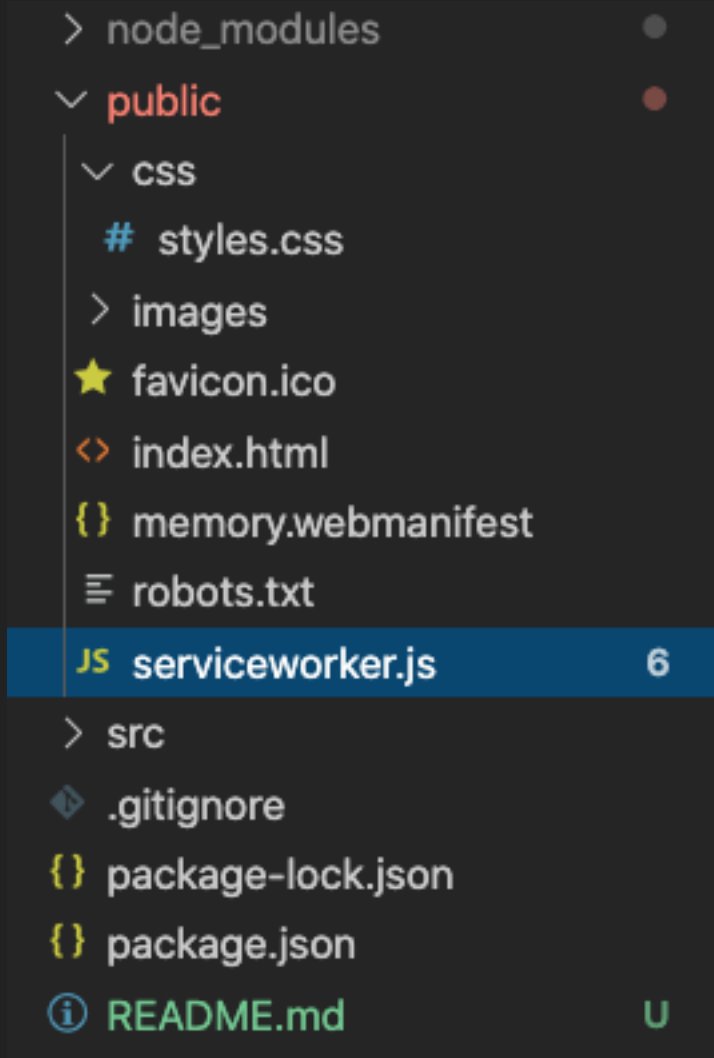


- ✓ DOM
- ✓ WebStorage

A ServiceWorkers lifecycle



Step 1 – Register the Service Worker



The service worker will be registered in a scope. It will only be available in that scope.

To get scope for ./, you need to place the serviceworker-file in that folder.

Step 1 – Register the Service Worker

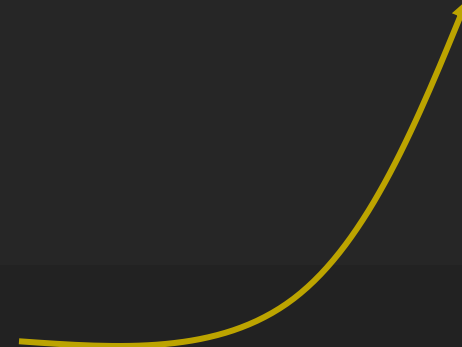
“Feature detection”
Does this browser support service worker?

Make sure site is loaded before adding the service worker. Should not be necessary when using ESM

```
if ('serviceWorker' in navigator) {  
  window.addEventListener('load', async () => {  
    try {  
      const registration = await navigator.serviceWorker.register('./serviceworker.js')  
  
      console.log('ServiceWorker: Registration successful with scope: ', registration.scope)  
    } catch (error) {  
      console.log('ServiceWorker: Registration failed: ', error)  
    }  
  })  
}
```

src/index.js

http://localhost:8080/
https://mysite.se/



serviceworker.js

```
const version = '1.0.0'

self.addEventListener('install', event => {
  console.log('ServiceWorker: Installed version ', version)
  // TODO: Cache resources needed to start
})

self.addEventListener('activate', event => {
  console.log('ServiceWorker: Activated version', version)
  // TODO: Clean up older versions of the cache
})

self.addEventListener('fetch', event => {
  console.log('ServiceWorker: Fetching')
  // TODO: Cache new resources when online and serve cached content if offline
})

self.addEventListener('message', event => {
  console.log('ServiceWorker: Got a message')
  // TODO: Handle events from the main application
})

self.addEventListener('push', event => {
  console.log('ServiceWorker: Got a push message from the server')
  // TODO: Show a notification for the user
})
```

CacheStorage

Stores Request (key) and Response (value) objects

The screenshot shows the Chrome DevTools interface. The left sidebar has tabs for Elements, Console, Sources, Network, Performance, Memory, Application, Security, Lighthouse, and Layers. The Application tab is active, showing a tree view with sections: Application (Manifest, Service Workers, Clear storage), Storage (Local Storage, Session Storage, IndexedDB, Web SQL, Cookies), Cache (Cache Storage), and Background Services (Background Fetch, Background Sync). Under Cache Storage, the entry '2.0.0 - http://localhost:8080' is selected. The right pane shows the Network tab with a table of resources. The table has columns: #, Name, Response-Type, and Content-Type. The selected resource is at index 11, with Name '/js/components/my-memory-game/images/1.png', Response-Type 'basic', and Content-Type 'image/png'. Below the table, the 'Preview' tab is active, showing the content of the selected resource as a JavaScript code snippet.

#	Name	Response-Type	Content-Type
0	/	basic	text/html
1	/__snowpack__/env.js	basic	application/ja...
2	/__snowpack__/hmr-client.js	basic	application/ja...
3	/__snowpack__/hmr-error-overlay.js	basic	application/ja...
4	/css/styles.css	basic	text/css
5	/favicon.ico	basic	image/vnd.mi...
6	/images/homescreen192.png	basic	image/png
7	/index.html	basic	text/html
8	/js/components/my-flipping-tile/index.js	basic	application/ja...
9	/js/components/my-flipping-tile/my-flipping-tile.js	basic	application/ja...
10	/js/components/my-memory-game/images/0.png	basic	image/png
11	/js/components/my-memory-game/images/1.png	basic	image/png
12	/js/components/my-memory-game/images/2.png	basic	image/png

```
// It is best practice to have a index.js as root of the component. You can add the conte
import './my-flipping-tile.js'
```

Line 1. Column 1

Caching on install example

```
self.addEventListener('install', event => {
  console.info('ServiceWorker: Installed version ', version)

  /**
   * Cache assets when installing the service worker.
   *
   * @returns {Promise} Promise that resolves to undefined.
   */
  const cacheAssets = async () => {
    const cache = await self.caches.open(version)

    console.log('ServiceWorker: Caching Files')

    return cache.addAll([
      'index.html',
      'css/styles.css'
    ])
  }

  // Make sure to wait in "installing phase" until assets are cached.
  event.waitUntil(cacheAssets())
})
```

Caching on fetch

```
const cachedFetch = async request => {  
  try {  
    // Try to fetch the asset from the server and if success, clone the result.  
    const response = await fetch(request)  
  
    // Save the result in the cache  
    const cache = await self.caches.open(version)  
    cache.put(request, response.clone())  
  
    return response  
  } catch (error) {  
    console.info('ServiceWorker: Serving cached result')  
    return self.caches.match(request)  
  }  
}  
  
self.addEventListener('fetch', event => {  
  console.info('ServiceWorker: Fetching')  
  
  event.respondWith(cachedFetch(event.request))  
})
```

Need to clone since we do not wait for the cache to save the response, before we return.

Remove Old Cached Assets

```
const removeCachedAssets = async () => {  
  const cacheKeys = await self.caches.keys()  
  
  return Promise.all(  
    cacheKeys.map(cache => {  
      if (cache !== version) {  
        console.info('ServiceWorker: Clearing Cache', cache)  
        return self.caches.delete(cache)  
      }  
    })  
  )  
}  
  
self.addEventListener('activate', event => {  
  console.log('ServiceWorker: Activated version', version)  
  
  event.waitUntil(removeCachedAssets())  
})
```

Application/ServiceWorkers

The screenshot shows the Chrome DevTools Application tab. The left sidebar contains a tree view with the following items: **Application** (selected), **Manifest**, **Service Workers** (highlighted), **Clear storage**, **Storage** (expanded), **Local Storage**, **Session Storage**, **IndexedDB**, **Web SQL**, **Cookies**, and **Cache**. The main panel displays the **Service Workers** section for the URL **http://localhost:8080/**. It includes three toggle options: **Offline** (unchecked), **Update on reload** (checked), and **Bypass for network** (unchecked). Below these, there are links for **Update** and **Unregister**. The **Source** is listed as **serviceworker.js**, and the **Received** timestamp is **30/11/2020, 14:32:28**. The **Status** is **#748 activated and is running**, with a **stop** link. The **Clients** section shows **http://localhost:8080/** with a **focus** link. At the bottom, there is a **Push** button and a text input field containing **Test push message from DevTools**.

Application

- Manifest
- Service Workers**
- Clear storage

Storage

- Local Storage
- Session Storage
- IndexedDB
- Web SQL
- Cookies

Cache

Service Workers

☐ Offline ☒ Update on reload ☐ Bypass for network

http://localhost:8080/ [Update](#) [Unregister](#)

Source [serviceworker.js](#)

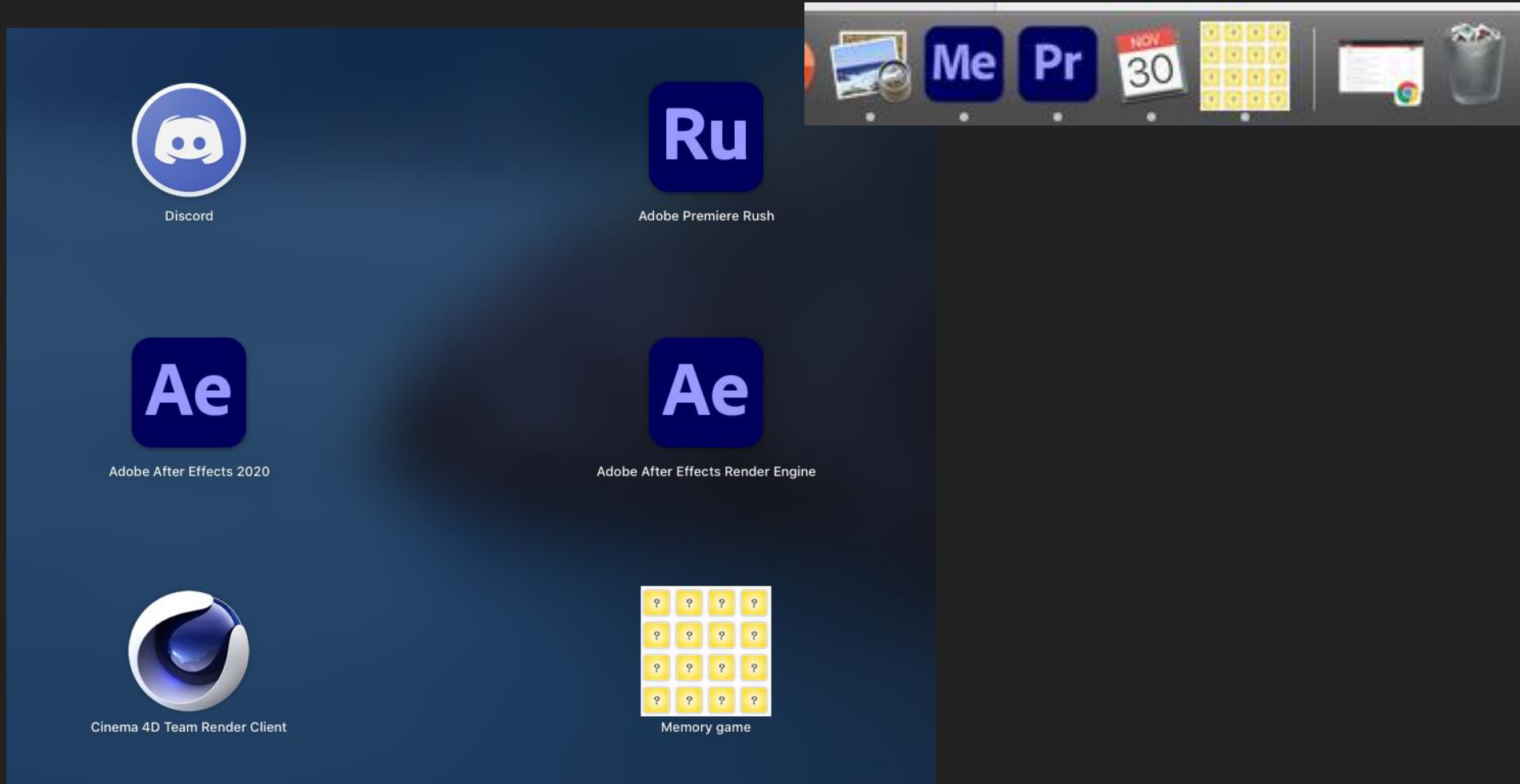
Received 30/11/2020, 14:32:28

Status ● #748 activated and is running [stop](#)

Clients [http://localhost:8080/](#) [focus](#)

Push [Push](#)

Install your PWA as an application



Web Manifest

- ✓ Requirements:
 - ✓ A manifest file
 - ✓ HTTPs
 - ✓ Icon
 - ✓ A registered service worker (Chromium only)

.webmanifest

```
{
  "name": "Memory game",
  "short_name": "Memory",
  "start_url": "./",
  "display": "standalone",
  "background_color": "#ffe001",
  "description": "A simple memory game!",
  "icons": [{
    "src": "images/homescreen48.png",
    "sizes": "48x48",
    "type": "image/png"
  }, {
    "src": "images/homescreen192.png",
    "sizes": "192x192",
    "type": "image/png"
  }]
}
```

memory.webmanifest

App name

- Application type:
- fullscreen
 - standalone
 - minimal-ui
 - browser

```
<link rel="manifest"
href="/memory.webmanifest" />
```

index.html

Application

Manifest

Service Workers

Clear storage

Storage

Local Storage

Session Storage

IndexedDB

Web SQL

Cookies

Cache

Cache Storage

2.0.0 - http://localhost:8080

Application Cache

App Manifest

memory.webmanifest

Identity

Name	Memory game
Short name	Memory

Presentation

Start URL	./
Theme color	
Background color	■ #ffe001
Orientation	

<https://web.dev/add-manifest/>

Some things to note in our environment

- ✓ Make sure to put the manifestfile and the serviceworker in the “public”-folder.
- ✓ Make sure to toggle on “Update on reload” at the service worker in the inspector
- ✓ Make sure to toggle on “Perserve log” on

